OtterSec

Picasso Network

Security Assessment

April 23rd, 2024 — Prepared by OtterSec

Akash Gurugunti sud0u53r.ak@osec.io

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Table of Contents

01 — Executive Summary

Overview

Picasso Network engaged OtterSec to assess the `emulated-light-client` program. This assessment was conducted between February 6th and April 19th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 11 findings throughout this audit engagement.

In particular, we identified a critical vulnerability, concerning the potential lockup of funds during withdrawal due to a lack of compatibility for handling cases where no value is passed to the optional service parameter (OS-CFI-ADV-05) and a high-risk issue enabling unauthorized alterations to staking parameters (OS-CFI-ADV-00). Furthermore, we highlighted the lack of account validation in the deposit and staking functionalities (OS-CFI-ADV-01).

We also made suggestions regarding consistency in code documentation and comments describing the actual functionalities and the usage of proper error messages (OS-CFI-SUG-04). Additionally, we recommended the removal of redundant code (OS-CFI-SUG-02) and advised certain optimizations to improve the overall efficiency of the system (OS-CFI-SUG-03).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/ComposableFi/emulated-light-client. This audit was performed against commit ae55a30. We conducted a follow-up review on the commit 5d479f1.

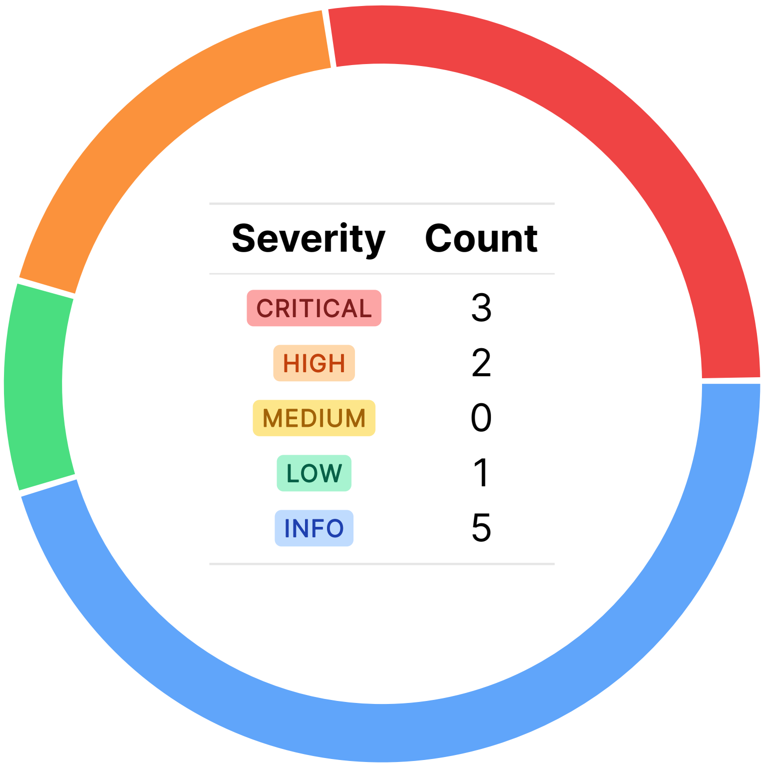
A brief description of the programs is as follows:

Name	Description
emulated-light-client	The module describes a bridge between Solana and Cosmos using Inter-Blockchain Communication (IBC).

03 — Findings

Overall, we reported 11 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	3
HIGH	2
MEDIUM	0
LOW	1
INFO	5

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-CFI-ADV-00	CRITICAL	RESOLVED ✓	`initialize` uses `init_if_needed` , allowing unauthorized alterations to staking parameters.
OS-CFI-ADV-01	CRITICAL	RESOLVED ✓	Lack of account validation for `remaining_accounts` and missing parameters for identifying the specific mint of the staked amount.
OS-CFI-ADV-02	CRITICAL	RESOLVED ✓	Absence of a check for a non-zero balance in the depositor's `receipt_token_account` within `set_service` instruction.
OS-CFI-ADV-03	HIGH	RESOLVED ✓	The lack of validation for the `instruction` sysvar account in `validate_remaining_accounts` and `set_stake` may lead to unintended or insecure usage.
OS-CFI-ADV-04	HIGH	RESOLVED ✓	`withdrawal_request` fails to update the `last_received_rewards_height` parameter, which may result in inaccurate reward calculations if a withdrawal request is canceled and raised again.
OS-CFI-ADV-05	LOW	RESOLVED ✓	There is a possibility of locking funds during withdrawal due to a lack of compatibility with `None` values for the optional `service` parameter.

Ability To Initialize Multiple Times CRITICAL OS-CFI-ADV-00

Description

In the `Initialize` instruction, while initializing the staking parameters, due to the use of `init_if_needed`, the staking parameters may be altered with new values multiple times by anyone. The ability to initialize the staking parameters multiple times may result in security vulnerabilities. For example, an attacker may repeatedly call the `Initialize` instruction with different parameters, altering the staking configuration and affecting the entire protocol.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

    #[account(init_if_needed, payer = admin, seeds = [STAKING_PARAMS_SEED, TEST_SEED], bump,
        ↪  space = 1024)]
    pub staking_params: Account<'info, StakingParams>,

    pub rewards_token_mint: Account<'info, Mint>,
    #[account(init_if_needed, payer = admin, seeds = [REWARDS_SEED, TEST_SEED], bump,
        ↪  token::mint = rewards_token_mint, token::authority = staking_params)]
    pub rewards_token_account: Account<'info, TokenAccount>,
    [...]
}
```

Remediation

Use `init` instead of `init_if_needed` for the `Initialize` instruction. This ensures that the initialization can only happen once.

Patch

Fixed by using `init` instead of `init_if_needed` for `Initialize` in e565006.

Discrepancies In Deposit Functionality CRITICAL OS-CFI-ADV-01

Description

`deposit` uses `remaining_accounts` for the `CPI` call to the guest chain program (`solana_ibc::cpi::set_stake`). However, the function lacks explicit validation checks on the `remaining_accounts` in `deposit` instruction. Similarly, the `solana_ibc::cpi::set_stake` function also lacks explicit validation checks for the accounts passed in the `CpiContext`.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
    // Call Guest chain program to update the stake if the chain is initialized
    if guest_chain_program_id.is_some() {
        [...]
        let cpi_program = ctx.remaining_accounts[3].clone();
        let cpi_ctx =
            CpiContext::new_with_signer(cpi_program, cpi_accounts, seeds);
        solana_ibc::cpi::set_stake(cpi_ctx, amount as u128)?;
    }
    Ok(())
}
```

Additionally, on invoking `solana_ibc::cpi::set_stake`, it is crucial to include parameters that identify the specific mint of the staked amount. Tokens on Solana may have different decimal places, and each mint may have a different scale. Without passing information about the mint of the staked amount, there is a risk of updating the stake value with an incorrect scale.

Remediation

Add validation checks in both `deposit` and `solana_ibc::cpi::set_stake` to ensure that the required accounts are present and have the correct ownership and include the mint information as a parameter when calling `set_stake`.

Patch

Fixed by adding validation checks to the `remaining_accounts` in b7847d9 and by asserting decimals of the `token_mint` to be 9 in 8b24f28.

Missing Receipt Token Balance Check CRITICAL OS-CFI-ADV-02

Description

In the implementation of `set_service` instruction, there is a section of code that sets the service for the stake which was deposited before guest chain initialization without explicitly checking if the depositor's `receipt_token_account` has a non-zero balance. The code assumes that the depositor has a sufficient balance in their `receipt_token_account` to cover the stake, but it fails to check for it explicitly.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn set_service<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, SetService<'info>>,
    service: Service,
) -> Result<()> {
    [...]
    vault_params.service = Some(service);
    let guest_chain_program_id =
        staking_params.guest_chain_program_id.unwrap(); // Infallible
    let amount = vault_params.stake_amount;
    [...]
}
```

Proof of Concept

- A malicious user sets an arbitrary service for a genuine user's `vault_params` by calling the `set_service` instruction using the genuine user's `vault_params` and an arbitrary `Service`.
- Since the code does not check for a non-zero balance in the `receipt_token_account`, the malicious user may abuse the system by setting an unauthorized stake for themselves using the original depositor's `vault_params`.

Remediation

Explicitly check if the depositor's `receipt_token_account` has a non-zero balance before proceeding with the stake setting. This check ensures the depositor has access to their respective `vault_params`.

Patch

Fixed by checking if the depositor's `receipt_token_account` has a non-zero balance in e10222d.

Lack Of Instruction Sysvar Validation HIGH OS-CFI-ADV-03

Description

The `instruction` sysvar account is passed to both `deposit` and `set_service` instructions, but its validation is not performed in `validate_remaining_accounts` and `set_stake`. Thus, it may be possible to replace or manipulate the `ihstruction` sysvar account, they might be able to inject unauthorized instructions into the CPI calls.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
#[derive(Accounts)]
pub struct Deposit<'info> {
    #[account(mut)]
    pub depositor: Signer<'info>,
    [...]
    ///CHECK:
    pub instruction: AccountInfo<'info>,
    [...]
}

#[derive(Accounts)]
pub struct SetService<'info> {
    #[account(mut)]
    depositor: Signer<'info>,
    [...]
    ///CHECK:
    pub instruction: AccountInfo<'info>,
    [...]
}
```

Remediation

Both the `validation::validate_remaining_accounts` and `set_stake` should include explicit validation for the `instruction` sysvar account. The validation should ensure that the account's address should match the expected value.

Patch

Fixed by checking the instruction `sysvar` account in b221448.

Inaccurate Reward Calculation HIGH OS-CFI-ADV-04

Description

In `withdrawal_request`, the `last_received_rewards_height` parameter of `vault_params` is not updated to the `current_height` after rewards are calculated and transferred.

>_ *restaking/programs/restaking/src/lib.rs* solidity

```
 pub fn withdrawal_request(ctx: Context<WithdrawalRequest>) -> Result<()> {
    let vault_params = &mut ctx.accounts.vault_params;
    let staking_params = &mut ctx.accounts.staking_params;
    let stake_token_mint = ctx.accounts.token_mint.key();
    [...]
    vault_params.withdrawal_request = Some(withdrawal_request_params);
    [...]
}
```

If a user cancels a withdrawal request and later requests withdrawal again, the function erroneously considers the `last_received_rewards_height` as the height when the last rewards were claimed, rather than the height when the last withdrawal request was raised. This affects the reward calculation as it will be based on outdated information.

Remediation

Update the `last_received_rewards_height` parameter to the current height after rewards are calculated and transferred within `withdrawal_request`.

Patch

Fixed by updating the `last_received_rewards_height` parameter accordingly in e69bba3.

Potential Fund Lockup LOW OS-CFI-ADV-05

Description

The `deposit` instruction includes an optional `service` parameter, which is of type `Option<Service>`. This parameter is used to specify a service associated with the staking operation. The vulnerability arises from the fact that the presence of the service parameter is later used as a condition during withdrawal. Specifically, the withdrawal logic includes a check on the service parameter.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
     vault_params.service =
            if guest_chain_program_id.is_some() { service } else { None };
    [...]
}
```

The code assumes that the service parameter will always be `Some(service)` during withdrawal. However, if the `deposit` instruction is called with `None` for the service parameter, this condition will not be met. Thus, If a deposit is made without specifying a service (i.e., `None` is passed), and the withdrawal logic assumes that there is always a service (`is_some()` condition), it may result in the lockup of funds.

Remediation

Modify `withdraw` to ensure compatibility with `None` values for the service parameter preventing fund lockup.

Patch

Fixed by adding instruction `set_service` to set the `service` parameter after depositing funds in 57edfe8.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-CFI-SUG-00	Proposal to replace `CpiContext::new_with_signer` with `CpiContext::new` in `set_stake`.
OS-CFI-SUG-01	Recommendation for changing `Option<Service>` parameter to `Service` in `deposit` to prevent users from setting `vault_params.service` to `None`.
OS-CFI-SUG-02	There are several instances of redundant or unnecessary code within the code base.
OS-CFI-SUG-03	optimizing `token::transfer` and `burn_nft` by introducing a boolean argument or using empty seeds to enable unsigned invocation in cases where signed invocation is unnecessary.
OS-CFI-SUG-04	Suggestions regarding consistency in code documentation and comments concerning the actual functionalities described by them and the usage of proper error messages.

Context Signer Correction OS-CFI-SUG-00

Description

The `CpiContext::new_with_signer` method is used to create the context for cross-program invocation. This method is typically used when a program expects a signed invocation, and it includes the account's seeds for signature verification.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Service,
    amount: u64,
) -> Result<()> {
    [...]
    let cpi_program = ctx.remaining_accounts[2].clone();
    let cpi_ctx =
        CpiContext::new_with_signer(cpi_program, cpi_accounts, seeds);
        solana_ibc::cpi::set_stake(cpi_ctx, validator_key, amount)?;
    [...]
}
```

Remediation

If `set_stake` does not require the `staking_params` account or its seeds for signature verification, using `CpiContext::new` is more appropriate and simplifies the context creation.

Enforce Mandatory Service Assignment

OS-CFI-SUG-01

Description

While handling the `vault_params.service` field in `deposit`, currently, it is defined as `Option<Service>`, allowing it to be either `Some(Service)` or `None`. The logic in the deposit function uses this option to conditionally set the `vault_params.service` based on `guest_chain_program_id.is_some`.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
}
```

However, the problem arises from the fact that even if `guest_chain_program_id` is `Some`, implying the guest chain is initialized, the service may still be set to `None`. This is due to the optionality of the `Service` type. Users may set `vault_params.service` to `None` even after the chain is initialized, undermining the intended behavior of the logic.

Remediation

Change the type of `vault_params.service` from `Option<Service>` to just `Service`. This modification ensures that a valid Service must always be provided once the guest chain is initialized.

Code Redundancy OS-CFI-SUG-02

Description

1. The `max_validators` value is currently stored in both the `Config` and `Candidates`. However, only the value of `max_validators` from `Candidates` is utilized. Therefore, the `max_validators` value in `Config` may be removed to reduce redundancy.
2. Within the `InitMint` structure, the `associated_token_program` field is not required and may be omitted as the `InitMint` operation does not directly interact with associated token accounts.

>_ *solana/solana-ibc/programs/solana-ibc/src/lib.rs* rust

```rust
#[derive(Accounts)]
#[instruction(port_id: ibc::PortId, channel_id_on_b: ibc::ChannelId, hashed_base_denom:
    ↪  CryptoHash)]
pub struct InitMint<'info> {
    #[account(mut)]
    sender: Signer<'info>,

    /// CHECK:
    #[account(init_if_needed, payer = sender, seeds = [MINT_ESCROW_SEED],
              bump, space = 100)]
    mint_authority: UncheckedAccount<'info>,

    #[account(init_if_needed, payer = sender, seeds = [hashed_base_denom.as_ref()],
              bump, mint::decimals = 6, mint::authority = mint_authority)]
    token_mint: Account<'info, Mint>,

    associated_token_program: Program<'info, AssociatedToken>,
    token_program: Program<'info, Token>,
    system_program: Program<'info, System>,
}
```

3. In `bit::fmt`, in the `else` branch, `off` is set to zero. Consequently, when `len` is decremented by `8 - off`, since `off` is already zero, `8 - off` will always equal eight. Therefore, in each iteration where `len` is already greater than or equal to eight, this operation is unnecessary and should be removed.

Remediation

Ensure that all redundant and unnecessary code is removed from the codebase.

Code Optimisation OS-CFI-SUG-03

Description

1. There is unnecessary signing and seed usage in certain `token::transfer` function calls. Specifically, in the `deposit` instruction, there are calls to `token::transfer` that do not require seeds or a signed invocation.
2. `handle` in `set` receives `NodeRef` (`nref`) as an argument. It extracts the pointer `nref.0` and hash `nref.1` from this structure, and then passes the entire `nref` to `handle_branch` and `handle_extension`, even though the hash (`nref.1`) is not needed in these calls.

>_ *common/sealable-trie/src/trie/set.rs* rust

```rust
fn handle(&mut self, nref: NodeRef) -> Result<(Ptr, CryptoHash)> {
    let nref = (nref.ptr.ok_or(Error::Sealed)?, nref.hash);
    [...]
    debug_assert_eq!(*nref.1, node.hash());
    match node {
        Node::Branch { children } => self.handle_branch(nref, children),
        Node::Extension { key, child } => {
            self.handle_extension(nref, key, child)
        }
    }
}
```

3. Within `get` in `trie`, `get_impl` is called with true for `include_proof` parameter, indicating that a proof should be included in the result, however, in the subsequent line, the proof is ignored, and only the value is retained. Including a proof in the `get_impl` call when it will not be used is inefficient.

Remediation

1. Modify `token::transfer` to handle both signed and unsigned invocations. This may be achieved by introducing a boolean argument or by checking the length of seeds.
2. Pass only the `Ptr` part (`nref.0`) to these functions, as the hash (`nref.1`) is not needed in those calls.
3. It would be more efficient to call `get_impl` with `include_proof` set to false if the proof is not intended to be used.

Code Maturity OS-CFI-SUG-04

Description

1. The documentation for the `RawNode` structure in `nodes`, states that `<key>` is a 36-byte array, however, it should be clarified that the actual bytes of the key extension are contained in a 34-byte array. This is because the first two bytes at the prefix are displayed separately in the binary representation, and are used to store information about the length of the key and the number of most significant bits to skip.

   >_ *common/sealable-trie/src/nodes.rs* rust

   ```rust
   // Extension: 1000_kkkk kkkk_kooo <key> <ref>
   //    `kkkk` is the length of the key in bits and `ooo` is number of most
   //    significant bits in <key> to skip before getting to the key.  <key> is
   //    36-byte array which holds the key extension.  Only `o..o+k` bits in it
   //    are the actual key; others are set to zero.
   ```

2. In `validation_context`, the error message returned by `get_packet_commitment` is incorrect. Instead of returning the `PacketCommitmentNotFound` error when the commitment is not found, it currently returns the `PacketReceiptNotFound` error.

   >_ */solana-ibc/src/validation_context.rs* rust

   ```rust
   fn get_packet_commitment(
       &self,
       path: &ibc::path::CommitmentPath,
   ) -> Result<ibc::PacketCommitment> {
       let trie_key = trie_ids::TrieKey::try_from(path)?;
       match self.borrow().provable.get(&trie_key).ok().flatten() {
           Some(hash) => Ok(hash.to_vec().into()),
           None => Err(ibc::ContextError::PacketError(
               ibc::PacketError::PacketReceiptNotFound {
                   sequence: path.sequence,
               },
           )),
       }
   }
   ```

3. Certain accounts are unnecessarily passed to instructions. Specifically, within `withdrawal_request` and `cancel_withdrawal_request`, the `nft_metadata` accounts is included. This account is not directly involved in the withdrawal or cancellation process.

Remediation

1. Update the documentation to reflect that the actual bytes of the key extension are contained in a 34-byte array, and the two bytes at the prefix are used for additional information about the key.
2. Ensure `get_packet_commitment` makes use of the `PacketCommitmentNotFound` error.
3. Remove the unnecessary account from the instruction accounts to streamline and optimize the code.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.